Projeto 1

Docker: Apresentação Teórica

Matheus Luis Oliveira da Silva

Opus Software

Teórica

O que é Docker?

- Ferramenta para gerenciamento e criação de containers;
- Criada em 2013 pela dotCloud, futura Docker;
- Motivada pela alta demanda por virtualização + inadequação gradativa das Maquinas Virtuais (VM’s) ao novo contexto.

História:

- Chroot 1979 → Alterar o diretório raiz de um processo (inicio do isolamento);
- FreeBSD Jails, LinuxVServer 2000 → Possibilidade de subdivisão do SO em segmentos menores com seu próprio hostname, diretório e endereço de rede. (início da virtualização “leve”);
- Solaris Containers Oracle 2004 → Inicio dos containers, particionando recursos e sistemas de arquivos e volumes;

História:

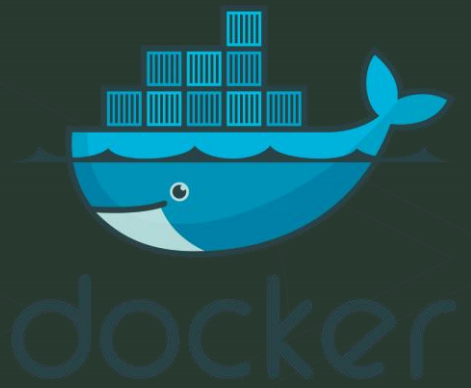
- Process Containers Google 2006 → Limitar, contabilizar e isolar uma arvore de processos, control groups, cgroups;
- LXC 2008 → primeira e mais completa ferramenta de contêinerização, utilizando de namespaces e cgroups, num unico kernel Linux;
- Warden cloud Foundry 2011 → Isolava ambientes em qualquer SO;
- dotCloud, Docker 2013 → Também começou com o LXC, mas posteriormente criou sua própria biblioteca, explodiu em popularidade.

Containerização para que, afinal?

- Compatibilidade e dependências;
- Tempo de configuração;
- Diferentes ambientes de uso.

Propostas e soluções:

- Independência entre componentes;
- Exclusividade de ambientes e dependências;
- Containers compartilham o mesmo Kernel do SO:
  - Compatibilidade 100%?



Open Container Initiative

- “The **Open Container Initiative** is an open governance structure for the express purpose of creating open industry standards around container formats and runtimes.”

Open Container Initiative

- Especificações para:
  - Runtimes (runtime-spec);
  - Imagens (image-spec);
  - Distribuição (distribution-spec).

Runtimes de Containers:

- Docker:
  - Docker swarm
- CRI-O:
  - Kubernetes
- PodMan:
  - DaemonLess
- Kata:
  - Virtual Machines
- PouchContainer:
  - AlibabaGroup

- Docker Swarm
- Kubernetes
- Marathon Mesos

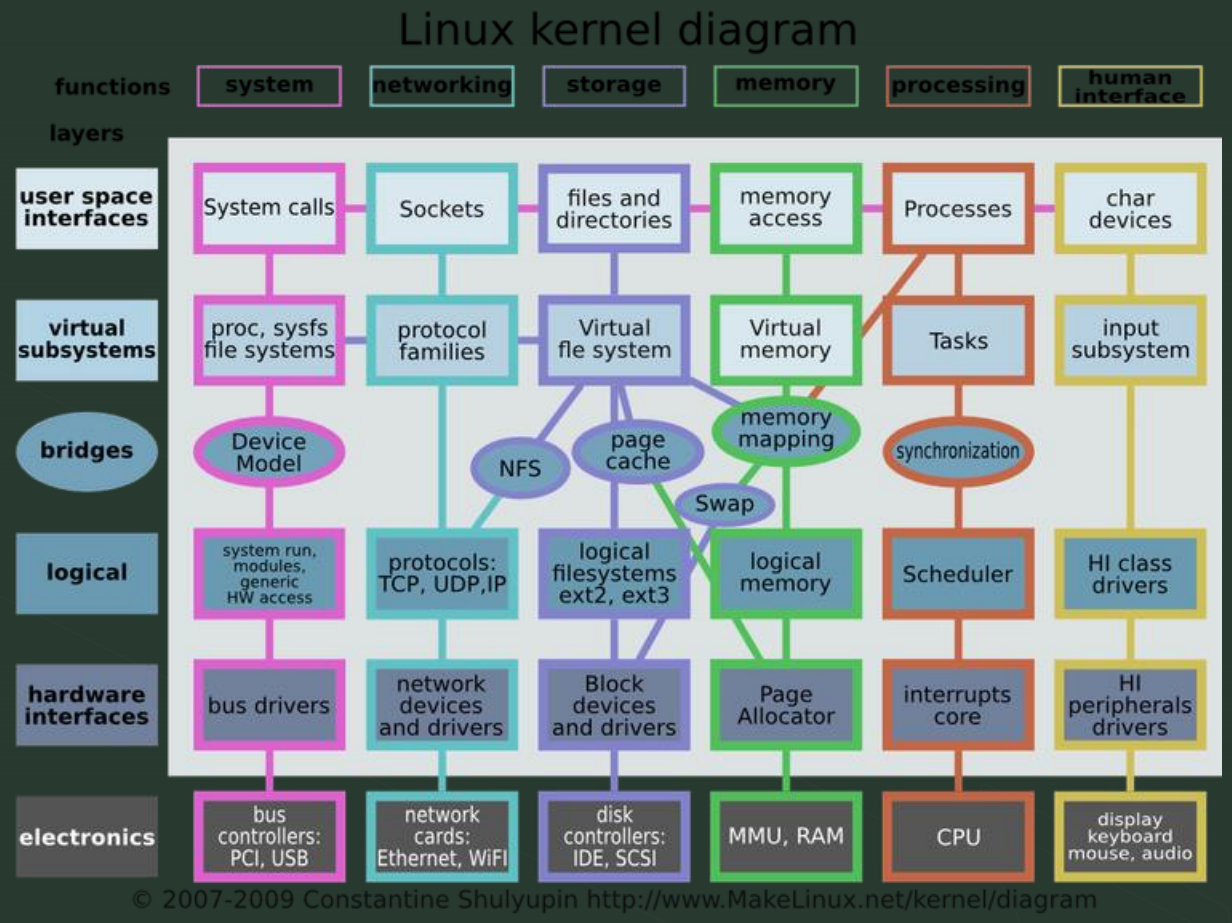
O que faz um runtime:

- Imagem é “puxada” de um registro para o local
- As camadas são extraídas com um sistema de copy-on-write, para preservá-las, bem como tendo para si um filesystem próprio, combinando-as.
- Os metadados definidos no build são implementados para preparação do container.
- O kernel é alertado para criação de um sistema isolado, processos, rede, sistema de arquivos, etc... Namespaces!
- O kernel então limita os recursos que esse sistema utilizará... Cgroups!
- Uma system call dispara o container

Pergunta de 1 milhão de dólares: Quais as diferenças entre VM’s e Containers?

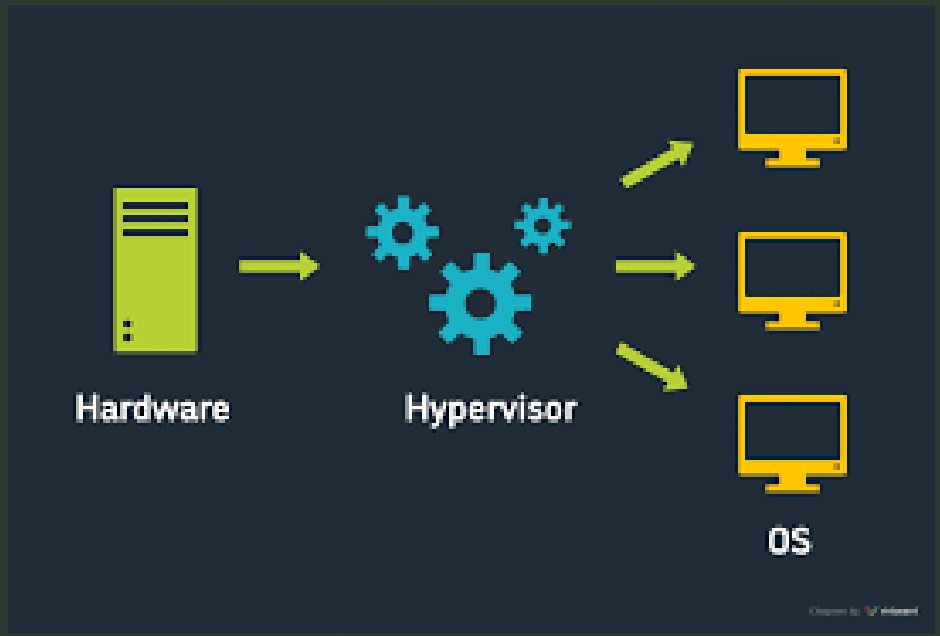
Revisão:

- Kernel



Revisão:

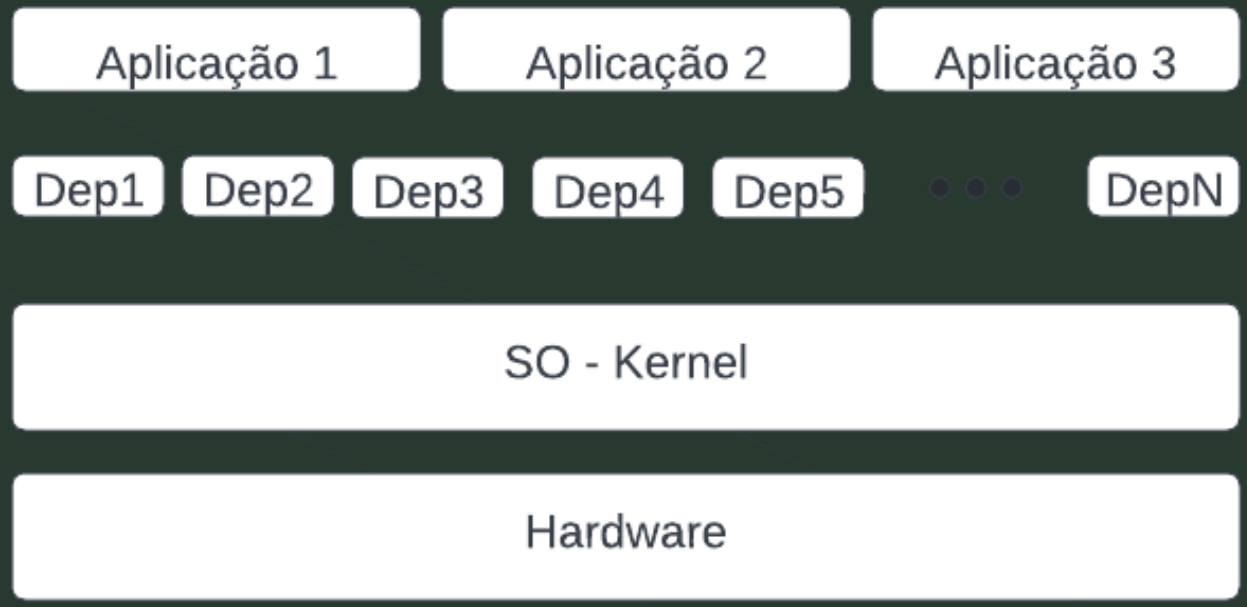
- Hypervisor
  - Tipo 1: ligação direta
  - Tipo 2: host dependency
  - Muitos privilégios de gerenciamento!



Revisão:

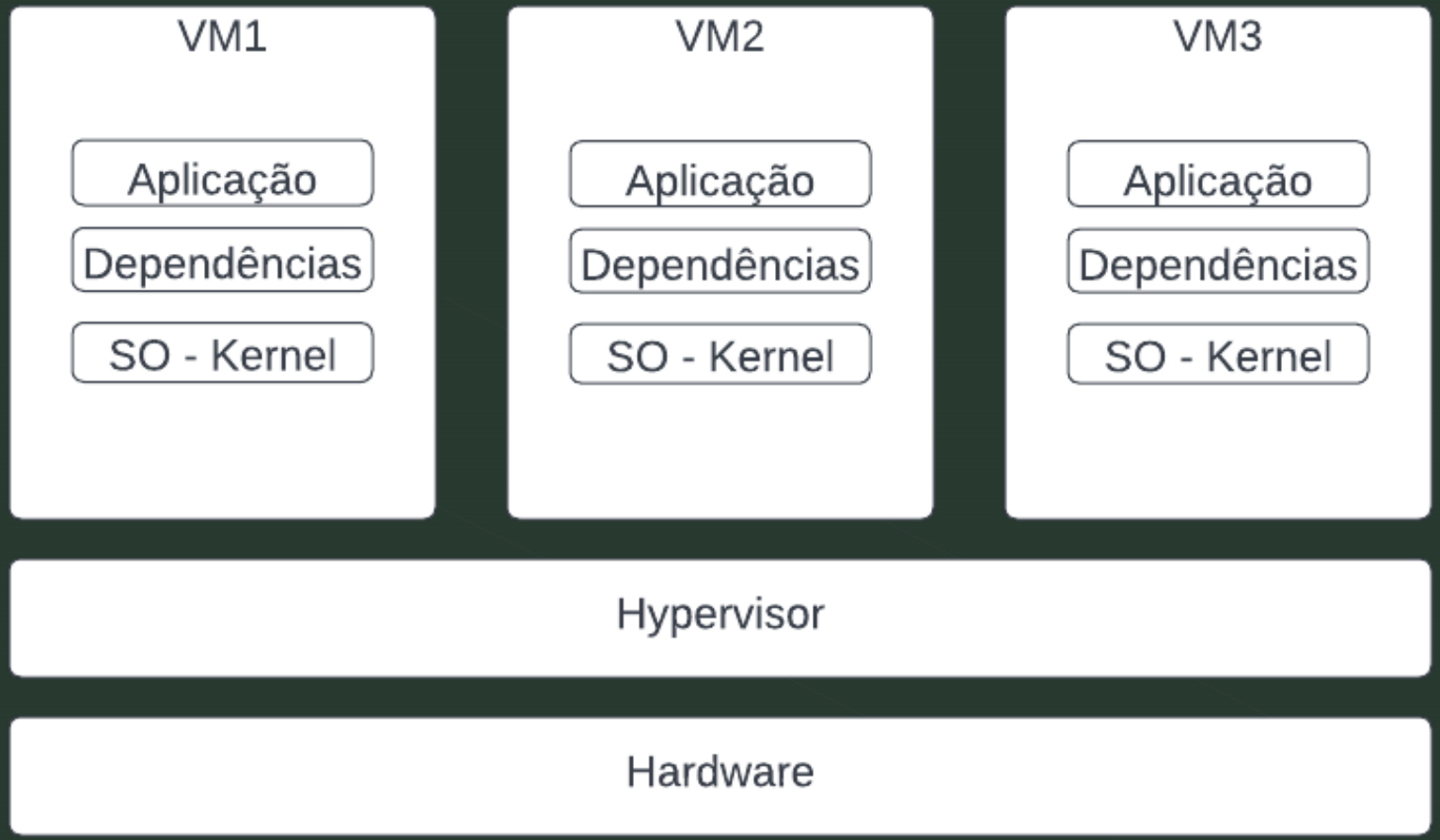
- SO

Revisão:



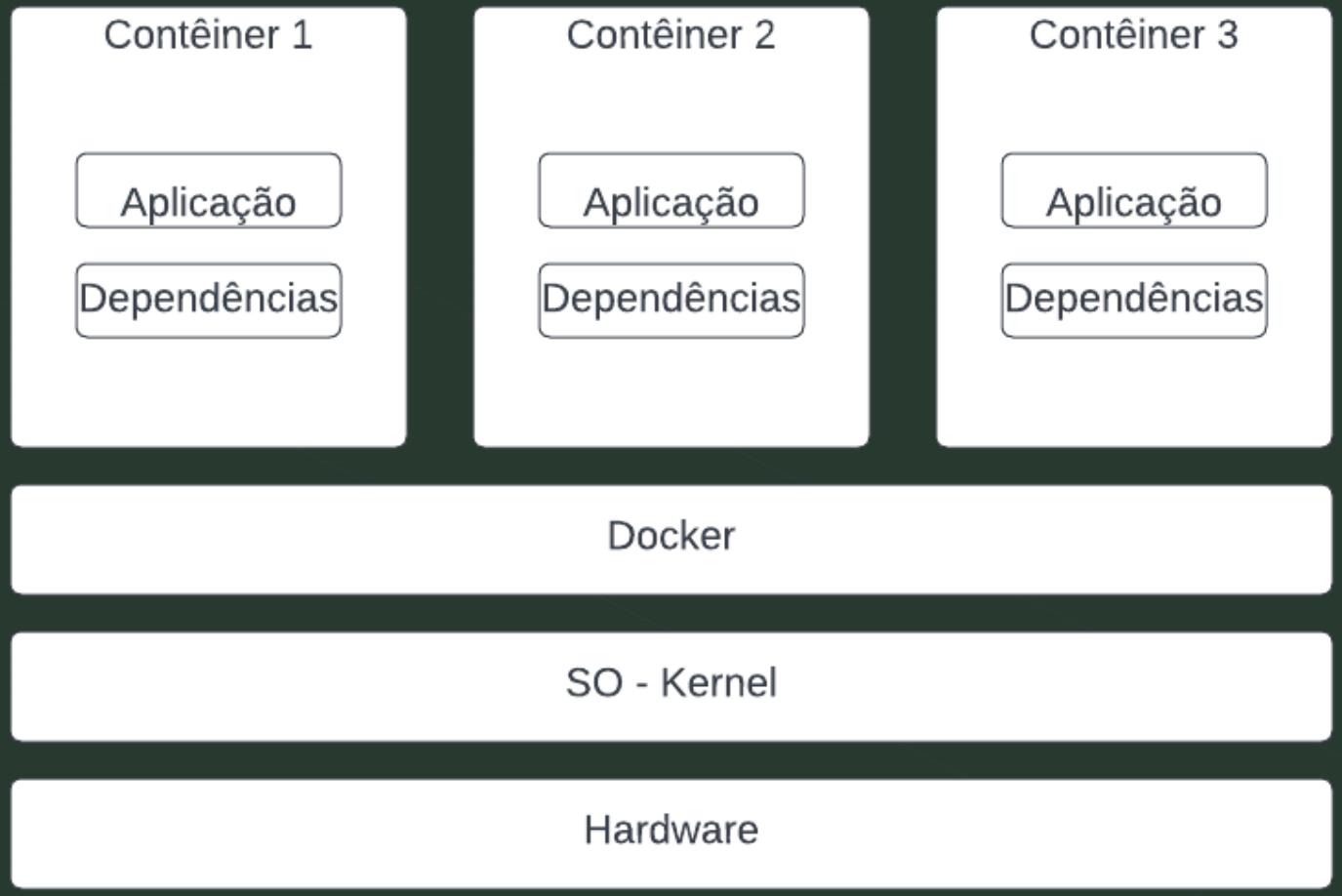
Próprio Autor

Revisão:



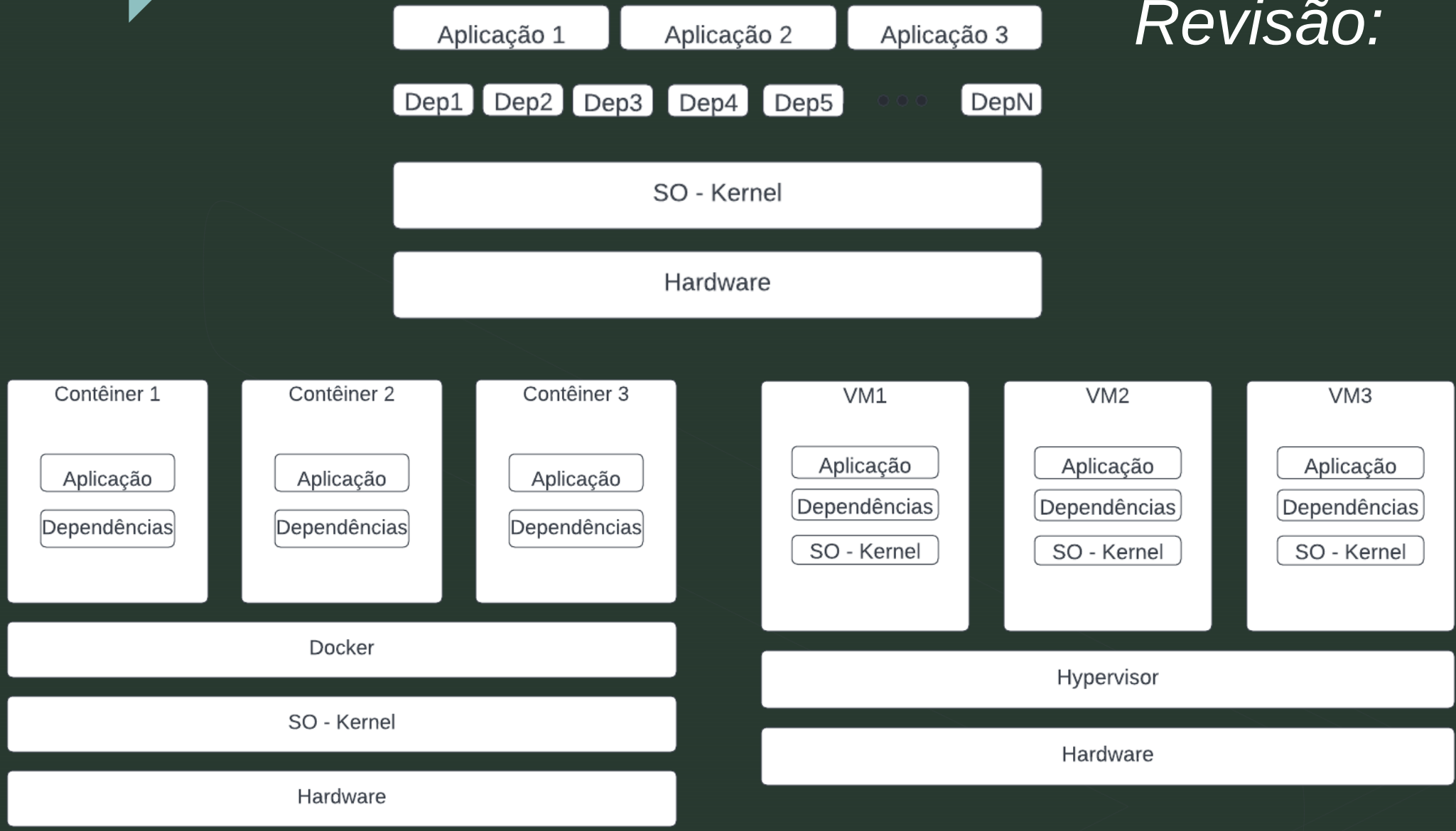
Próprio Autor

Revisão:



Próprio Autor

Revisão:


Aplicação 1
Aplicação 2
Aplicação 3
Dep1
Dep2
Dep3
Dep4
Dep5
DepN
SO - Kernel
Hardware
Contêiner 1
Aplicação
Dependências
Contêiner 2
Aplicação
Dependências
Contêiner 3
Aplicação
Dependências
Docker
SO - Kernel
Hardware
VM1
Aplicação
Dependências
SO - Kernel
VM2
Aplicação
Dependências
SO - Kernel
VM3
Aplicação
Dependências
SO - Kernel
Hypervisor
Hardware

Resumidamente:

- Maquinas Virtuais:
  - Usam quantidades significativas de recursos, dada a necessidade de virtualizar todo o SO + kernel para cada VM em execução.
  - Tem um tamanho em disco considerável, dado que as VM's são grandes e complexas.
  - Tempo de Inicialização alto, para subir todo o sistema operacional, suas dependências e softwares próprios.
- Contêineres:
  - Usam menos recursos, uma vez que que são mais leves, partindo do mesmo SO, virtualizam apenas o ambiente de desenvolvimento e processamento.
  - Usam menos espaço de disco, uma vez que são menores e possuem menos componentes.
  - Inicializam mais rápido, pelos mesmos motivos.
  - Entretanto, os contêineres necessitam do compartilhamento de recursos que não são necessários quando se utiliza uma VM, como o kernel, que no caso, possuem o seu próprio (resolvido pelo runtime do container)

Prós e Contras

- Prós:
  - Isolamento nível App
  - Boot mais rapido e leve
  - Mais fácil de gerenciar
  - Melhor sistema de versionamento
  - Alinhado com microsserviços

- Contras:
  - Muito novos! 2013 vs 1960
  - Poucos profissionais
  - Modismo? Ou tem robustez?
  - **Segurança!**

Segurança com docker

"Docker is about running random code downloaded from the Internet and running it as root.” -

https://opensource.com/business/14/7/docker-security-selinux

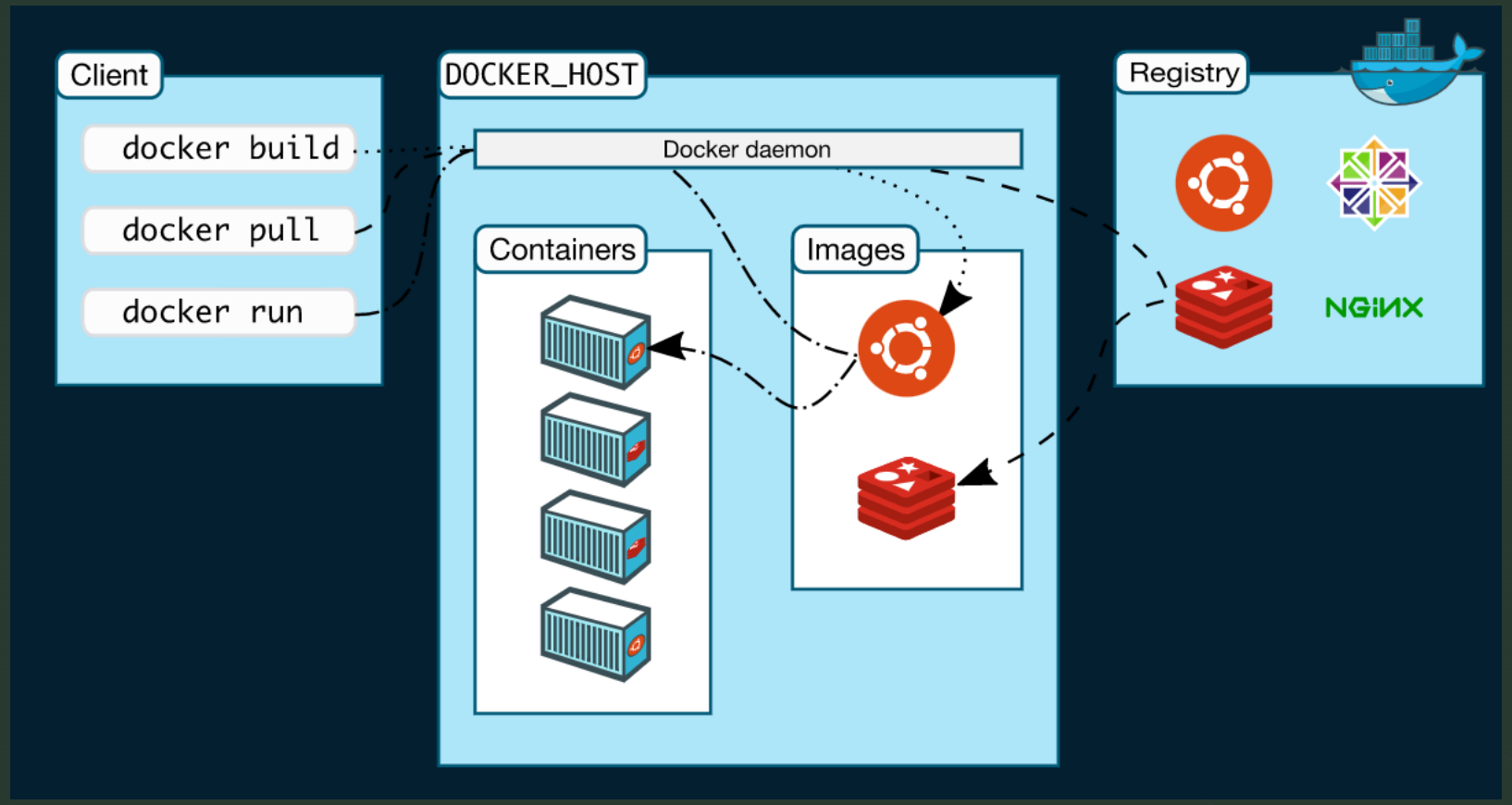
- São serviços como quaisquer outros!
- Privilégios devem ser tratados, assim como para qualquer serviço.
- Namespaces
- Networks
- Control Groups: DoS attack
- Cuidados com o docker daemon, ele está muito próximo!
  - Comparação com VM’s.

Docker

- LXC vs Docker
- Open-Source, Go
- Client-Server Architecture
  - REST:
    - Contrato de comunicação entre cliente e servidor
  - API:
    - Conjunto de restrições arquiteturais de comunicação entre softwares

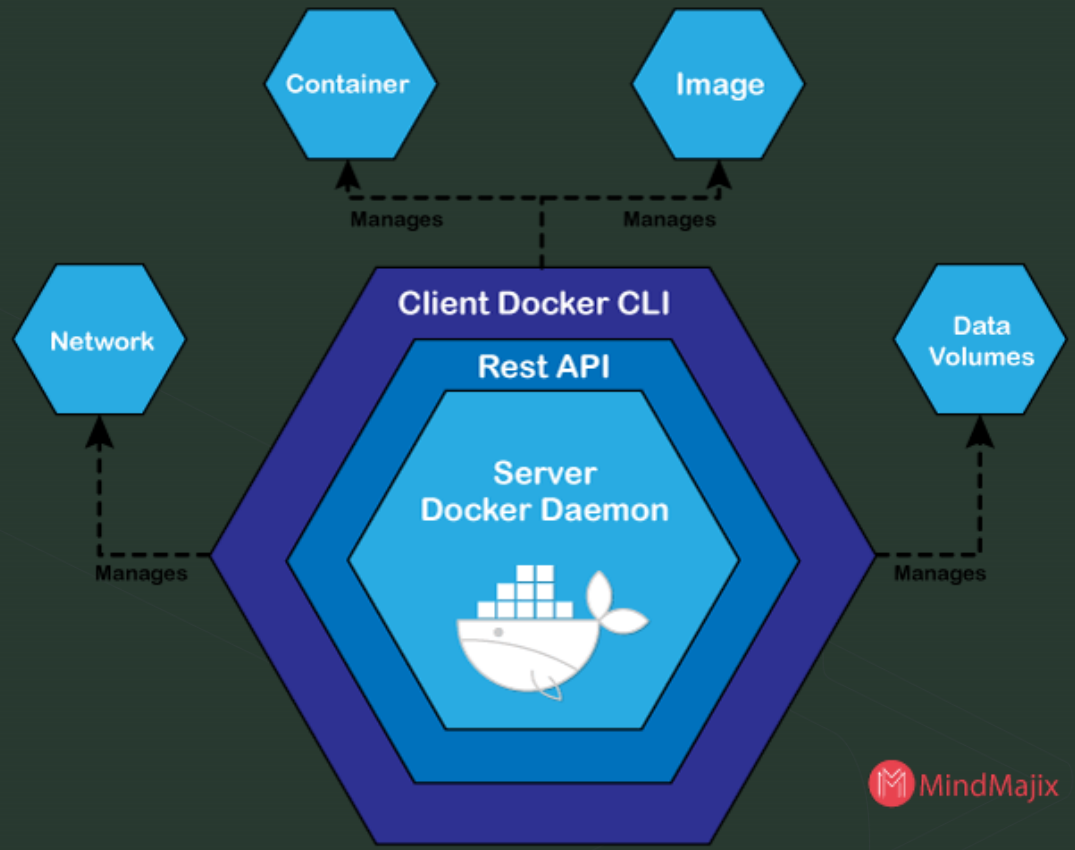
Docker

Arquitetura do docker...



Assim, temos seus principais componentes:

- Docker client
- Docker Daemon
- Docker Host
- Docker Registry
- Docker Objects:
  - Images
  - Containers
  - Storage
  - Network
  - ...



Images

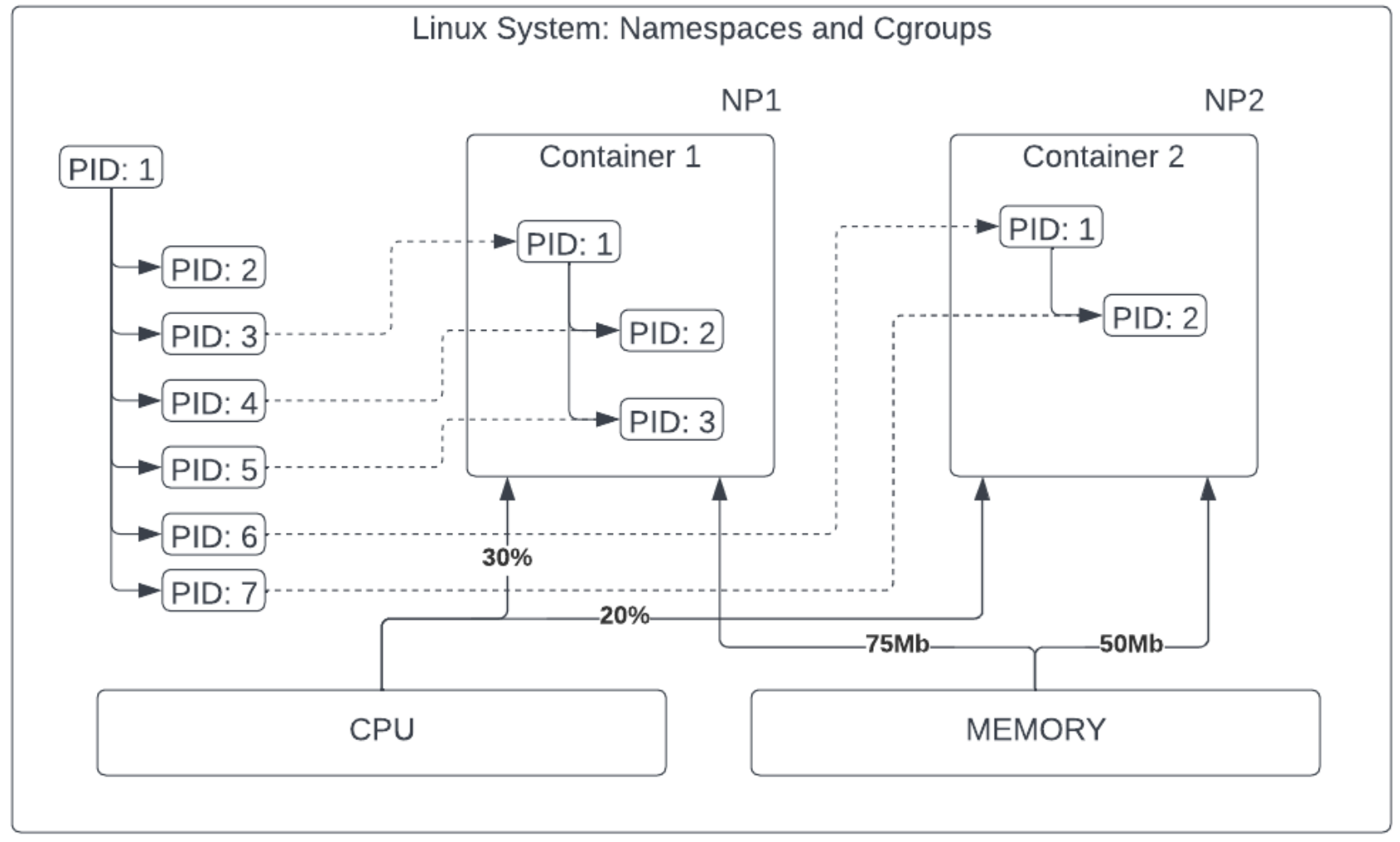
- Template com instruções para montagem do container
- DockerFile
- Camadas!

  - Cada instrução é uma layer;
  - Cada layer é uma imagem!?;
  - Uma layer somente aumenta o tamanho de uma imagem, o passado permanece – infelizmente o –tree foi desativado para mostrar
  - Imagens em comum são reutilizadas: cache.
- Cache
  - Docker possui uma cache que armazena todas as imagens já utilizadas;
  - Velocidade de build.
- Demonstração 1

Containers

- Instância de execução de uma imagem;
- Dependente de processos ativos:
  - Demonstração 2
- Isolamento de ambiente:
  - Demonstração 3
- Armazenamento efêmero:
  - Demonstração 4

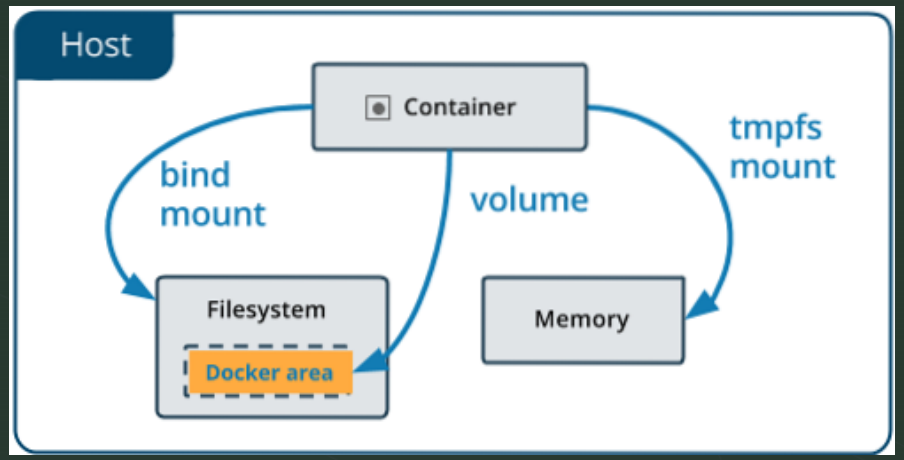
Isolamento dos containers:



Próprio Autor

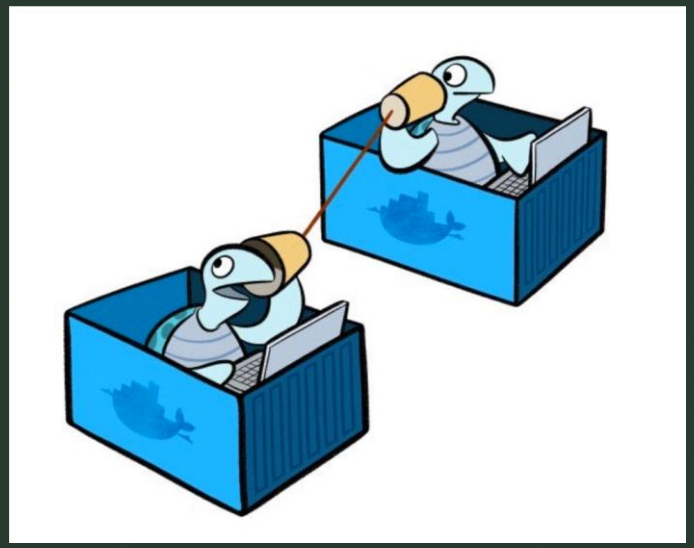
Docker Storage

- Armazenamento efêmero, a não ser que seja alocada uma unidade persistente.
- Camada Read-write, e procedimento copy-on-write.
  - Containers são, na sua maioria, leves!
- Data volumes:
  - instala uma unidade de armazenamento no filesystem do container conectado ao host.
  - Instalados no diretório /var/lib/docker/volumes do host
- Bind Mounts:
  - Qualquer lugar do host
- TMPFS:
  - Volume criado temporariamente na memoria do host



Network

- Bridge network:
  - Default bridge (!!!)
  - Conexão apenas entre participantes da bridge
  - Conexão apenas num mesmo host, distribuição de IP’s.
- User-Defined bridge:
  - Permite maior gerenciamento
  - Comunicação via nome (DNS)

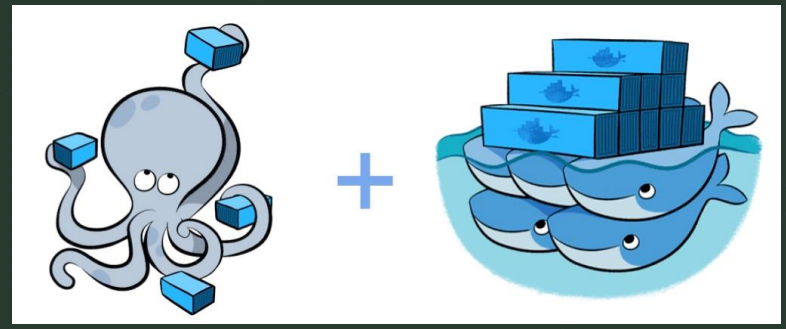


Network

- Host network:
  - O container se liga a rede do host;
  - Não permite portas identicas, containers passam a compartilhar o escopo.
- Overlay network:
  - Comunicação entre daemons diferentes.
- IPVlan MACvlan:
  - Endereços físicos ou lógicos
  - Interfaces de rede fisicas e lógicas emuladas
- Nenhuma rede.

Docker compose e Docker Swarm

- Compose:
  - Roda múltiplos containers com um só comando
  - Facilita a configuração da comunicação entre containers e mapeamentos de portas e volumes
  - Em suma, encapsula um conjunto de containers.
- Swarm:
  - Permite a interconexão de multiplos containers de diferentes daemons
  - Orquestrador de containers
  - Worker nodes e Manager nodes



Prática

Docker na prática:

- Baixar imagem: docker pull IMAGENAME
- DockerHub:
  - docker login –u USERNAME
  - docker tag USERNAME/IMAGETOCOPY:VERSION USERNAME/IMAGENAME:VERSION
  - docker push USERNAME/IMAGENAME:VERSION
- Build, commit: docker build -t USERNAME/IMAGENAME:VERSION CONTEXTDIRECTORY
- List: docker ps, docker network-or-volume ls, docker images, docker port
- Docker manager: docker run, stop, start
- Remove: docker rmi, rm, volume rm

Docker na prática:

- Exec: docker exec CONTAINER COMMAND
- Docker cp: docker cp ./some_file CONTAINER:/work
- Attatch, dettatch: docker attach CONTAINER, docker run IMAGE -d
- Inspect: docker inspect CONTAINER
- History: docker history IMAGE
- Logs: docker logs CONTAINER
- Create: volumes, networks, etc.

Docker Run:

- Run
  - --name = CONTAINERNAME
  - --network = none, host
  - --cpus=.5
  - --memory=100M
  - -d detach mode
  - -i permite stdin
  - -t permite um pseudo terminal
  - -p mapeamento de portas
  - -v mapeamento de volume
  - -e variáveis de ambiente

DockerFile:

- FROM → Imagem base da qual anova imagem será criada
- WORKDIR → Diretorio atual
- ARG → Variaveis de build da imagem
- ENV → Variaveis de ambiente para container
- EXPOSE → expor uma porta específica
- RUN → Comandos que serão executados no build da imagem
- COPY → Comando para copiar diretórios do sistema local para um diretorio dentro da imagem
- CMD → Comando que será executado quando a imagem for executada, substituído totalmente quando sobrescrito na chamada do docker
- ENTRYPOINT → Comando que será executado quando a imagem for executada, concatenado com a passagem de parametro na chamada do docker
- CMD + ENTRYPOINT → O CMD servirá como comportamento padrão
- Demonstração 5
- Demonstração 6

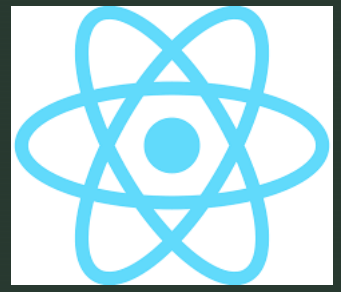
Docker-Compose:

- services:
  - CONTAINERNAME:
    - build:
    - image:
    - ports:
    - links or networks:
    - Volumes:
    - Environment:
    - configs:
    - depends_on:
- networks:
  - Net1
  - Net2
- Volumes:
  - V1

Docker-Compose:

- Docker compose up
- Docker compose down
- Docker compose ps

PROJETO PRÁTICO:
React + Node + Mysql + phpMyAdmin + Nginx



MySql

- Banco escolhido para a aplicação
- Conectado com a network DBNet
- Arquivo de inicialização do banco

Node

- Servidor que fará a conexão com o banco de dados
- Intermedia as requisições do frontend em react
- Conectado a rede serverNet e dbnet

PhpMyadmin

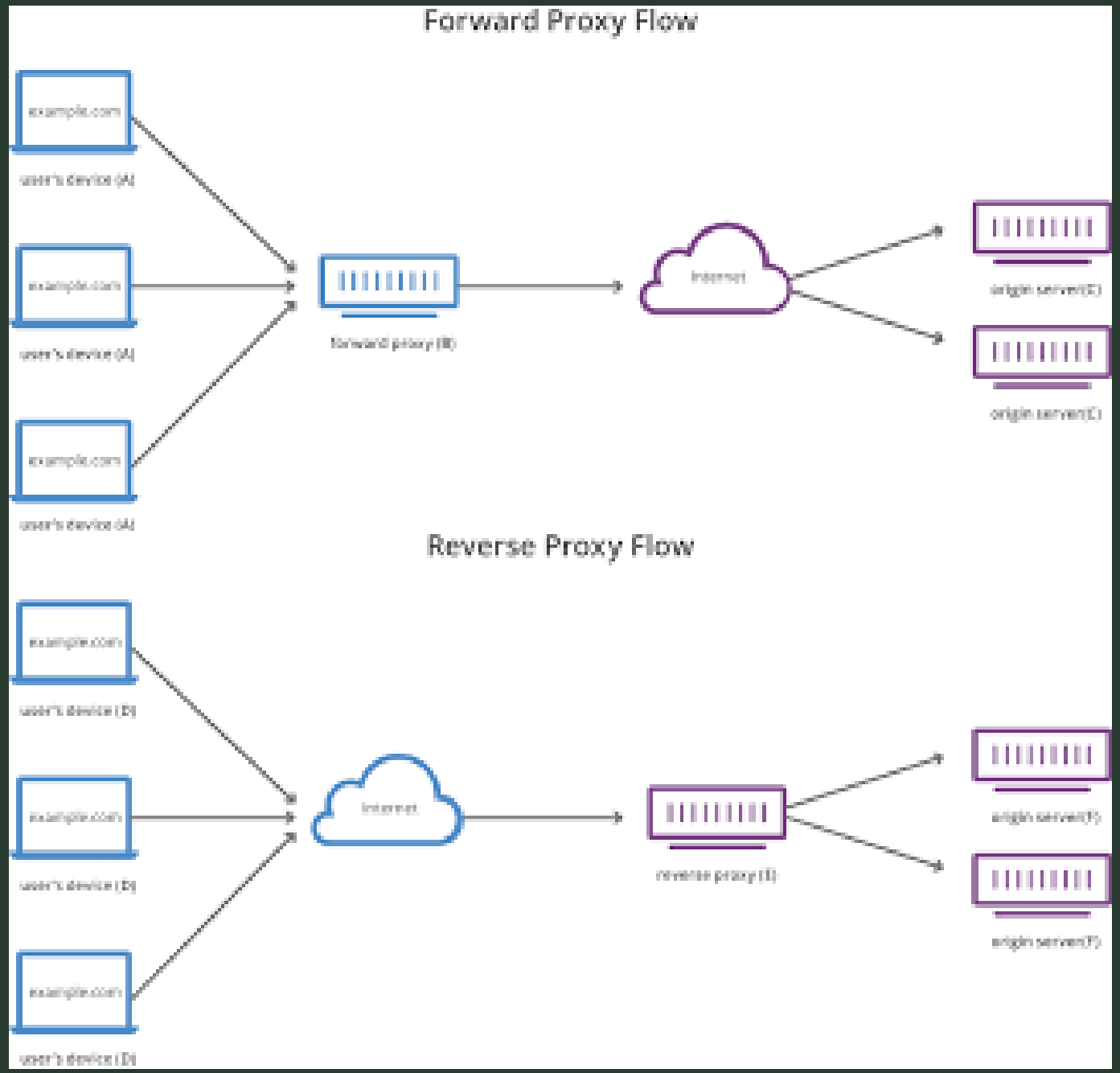
- Adminer do mysql para facilitar visualização e gerenciamento do banco
- Conectado a rede dbnet

React

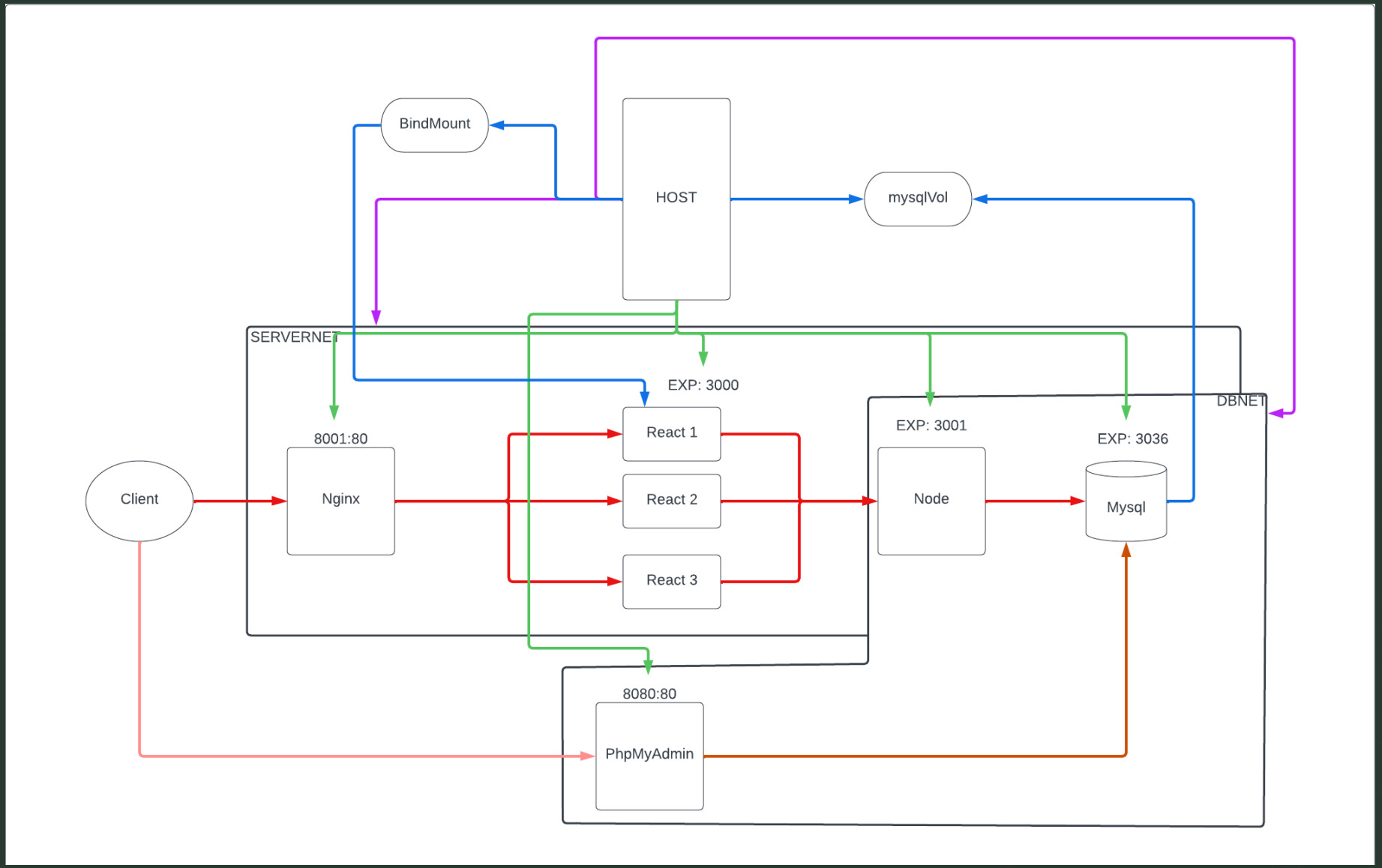
- Facilitador de desenvolvimento de frontend em javascript
- Conectado na rede servernet
- Tem sua requsiição ao banco intermediada pelo servidor node
- Não tem acesso ao banco

Nginx

- Por fim, o mais interessante, servidor proxy-reverse, com load balancing de fronts em react.
- Um exemplo simples dessas duas funcionalidades do nginx...
- Redirecionará requisições para três instancias do front em react...


Forward Proxy Flow
example.com
user's device (A)
example.com
user's device (A)
example.com
user's device (A)
forward proxy (B)
Internet
origin server(C)
origin server(C)
Reverse Proxy Flow
example.com
user's device (D)
example.com
user's device (D)
example.com
user's device (D)
Internet
reverse proxy (E)
origin server(F)
origin server(F)

Arquitetura



Referências para Imagens, respectivamente na ordem de aparição:

- https://www.pngwing.com/en/search?q=container
- http://webhosting24.ie/docker-hosting/
- https://www.pngitem.com/middle/hRTTwT_docker-container-logo-hd-png-download/
- https://opencontainers.org/
- https://makelinux.github.io/kernel/diagram/
- https://www.virtasant.com/blog/hypervisors-a-comprehensive-guide
- https://medium.com/hackernoon/introduction-to-operating-system-for-self-taught-web-developer-ba6d484398aa
- https://docs.docker.com/get-started/overview/
- https://mindmajix.com/what-is-docker-how-docker-works
- https://blog.eveo.com.br/armazenamento-definido-por-software
- https://pt.linuxteaching.com/article/storing_and_sharing_with_docker_volumes
- https://www.mundodocker.com.br/tag/rede-no-docker/
- https://codeblog.dotsandbrackets.com/docker-stack/